The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Computation Institute
University of Chicago

Workshop for Advancing Numerical Modeling
of Mantle Convection and Lithospheric Dynamics
UC Davis, CA July 9-11, 2008

Outline

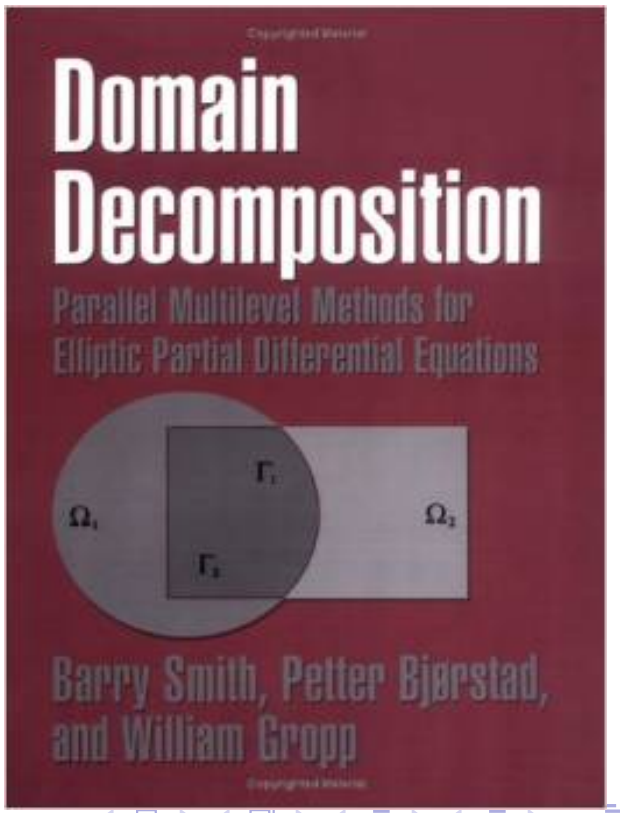
Outline

How did PETSc Originate?

PETSc was developed as a Platform for Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms
  - which blur these boundaries



The Role of PETSc

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

— Barry Smith

Advice from Bill Gropp

You want to think about how you decompose your data structures, how you think about them globally. [...] If you were building a house, you'd start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn't start with a bunch of tiles and say. "Well I'll put this tile down on the ground, and then I'll find a tile to go next to it." But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you're going to survive making your code parallel.

(http://www.rce-cast.com/Podcast/rce-28-mpich2.html)

What is PETSc?

A freely available and supported research code for the parallel solution of nonlinear algebraic equations

Free

- Download from http://www.mcs.anl.gov/petsc
- Free for everyone, including industrial users

Supported

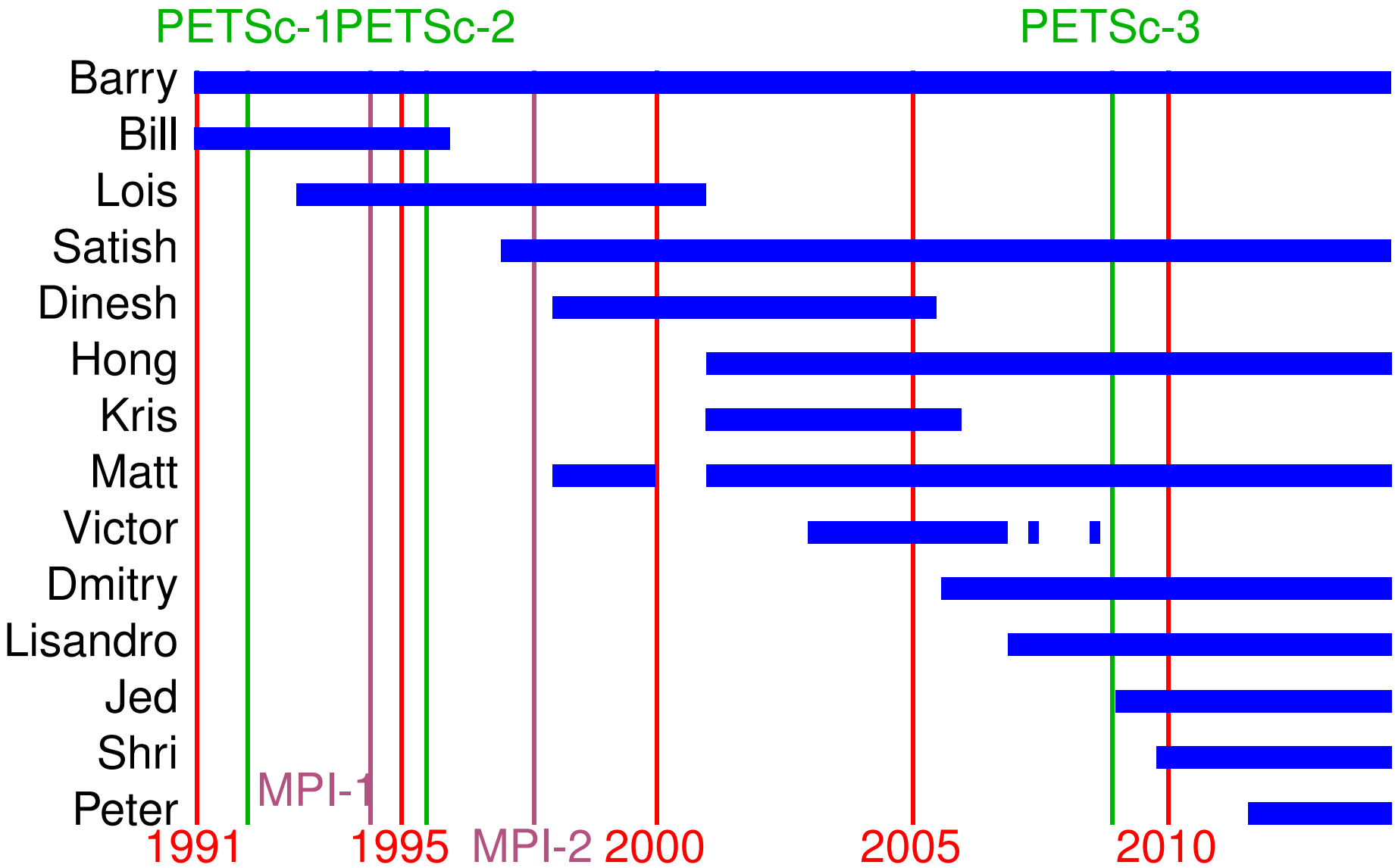
- Hyperlinked manual, examples, and manual pages for all routines
- Hundreds of tutorial-style examples
- Support via email: petsc-maint@mcs.anl.gov

Usable from C, C++, Fortran 77/90, Matlab, Julia, and Python

What is PETSc?

- Portable to any parallel system supporting MPI, including:
  - Tightly coupled systems
    - Cray XT6, BG/Q, NVIDIA Fermi, K Computer
  - Loosely coupled systems, such as networks of workstations
    - IBM, Mac, iPad/iPhone, PCs running Linux or Windows
- PETSc History
  - Begun September 1991
  - Over 60,000 downloads since 1995 (version 2)
  - Currently 400 per month
- PETSc Funding and Support
  - Department of Energy
    - SciDAC, MICS Program, AMR Program, INL Reactor Program
  - National Science Foundation
    - CIG, CISE, Multidisciplinary Challenge Program

Timeline


PETSc-1PETSc-2
PETSc-3
Barry
Bill
Lois
Satish
Dinesh
Hong
Kris
Matt
Victor
Dmitry
Lisandro
Jed
Shri
Peter
MPI-1
MPI-2
1991
1995
2000
2005
2010

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 290,000 cores efficiently
  - UNIC on the IBM BG/P Jugene at Jülich
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 290,000 cores efficiently
  - UNIC on the IBM BG/P Jugene at Jülich
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 290,000 cores efficiently
  - UNIC on the IBM BG/P Jugene at Jülich
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

Outline

Who Uses PETSc?

Computational Scientists

- Earth Science
  - PyLith (CIG)
  - Underworld (Monash)
  - Magma Dynamics (LDEO, Columbia, Oxford)
- Subsurface Flow and Porous Media
  - STOMP (DOE)
  - PFLOTRAN (DOE)

Who Uses PETSc?

Computational Scientists

- CFD
  - Firedrake
  - Fluidity
  - OpenFOAM
  - freeCFD
  - OpenFVM
- MicroMagnetics
  - MagPar
- Fusion
  - XGC
  - BOUT++
  - NIMROD

Who Uses PETSc?

Algorithm Developers

- Iterative methods
  - Deflated GMRES
  - LGMRES
  - QCG
  - SpecEst
- Preconditioning researchers
  - Prometheus (Adams)
  - ParPre (Eijkhout)
  - FETI-DP (Klawonn and Rheinbach)

Who Uses PETSc?

Algorithm Developers

- Finite Elements
  - libMesh
  - MOOSE
  - PETSc-FEM
  - Deal II
  - OOFEM
- Other Solvers
  - Fast Multipole Method (PetFMM)
  - Radial Basis Function Interpolation (PetRBF)
  - Eigensolvers (SLEPc)
  - Optimization (TAO)

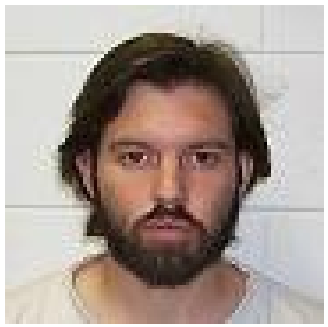
The PETSc Team


Bill Gropp


Barry Smith


Satish Balay


Jed Brown


Matt Knepley


Lisandro Dalcin


Hong Zhang


Mark Adams


Toby Issac

Outline

Downloading PETSc

- The latest tarball is on the PETSc site:
  http://www.mcs.anl.gov/petsc/download
- There is a Debian package (`aptitude install petsc-dev`)
- There is a Git development repository

Cloning PETSc

- The full development repository is open to the public
  - https://bitbucket.org/petsc/petsc/
- Why is this better?
  - You can clone to any release (or any specific ChangeSet)
  - You can easily rollback changes (or releases)
  - You can get fixes from us the same day
- All releases are just tags:
  - Source at tag v3.4.4

Automatic Downloads

- Starting in 2.2.1, some packages are automatically
  - Downloaded
  - Configured and Built (in `$PETSC_DIR/externalpackages`)
  - Installed with PETSc
- Currently works for
  - petsc4py
  - PETSc documentation utilities (Sowing, lgrind, c2html)
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, OpenMPI
  - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
  - MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
  - BLOPEX, FFTW, SPRNG
  - Prometheus, HYPRE, ML, SPAI
  - Sundials
  - Triangle, TetGen
  - FIAT, FFC, Generator
  - Boost

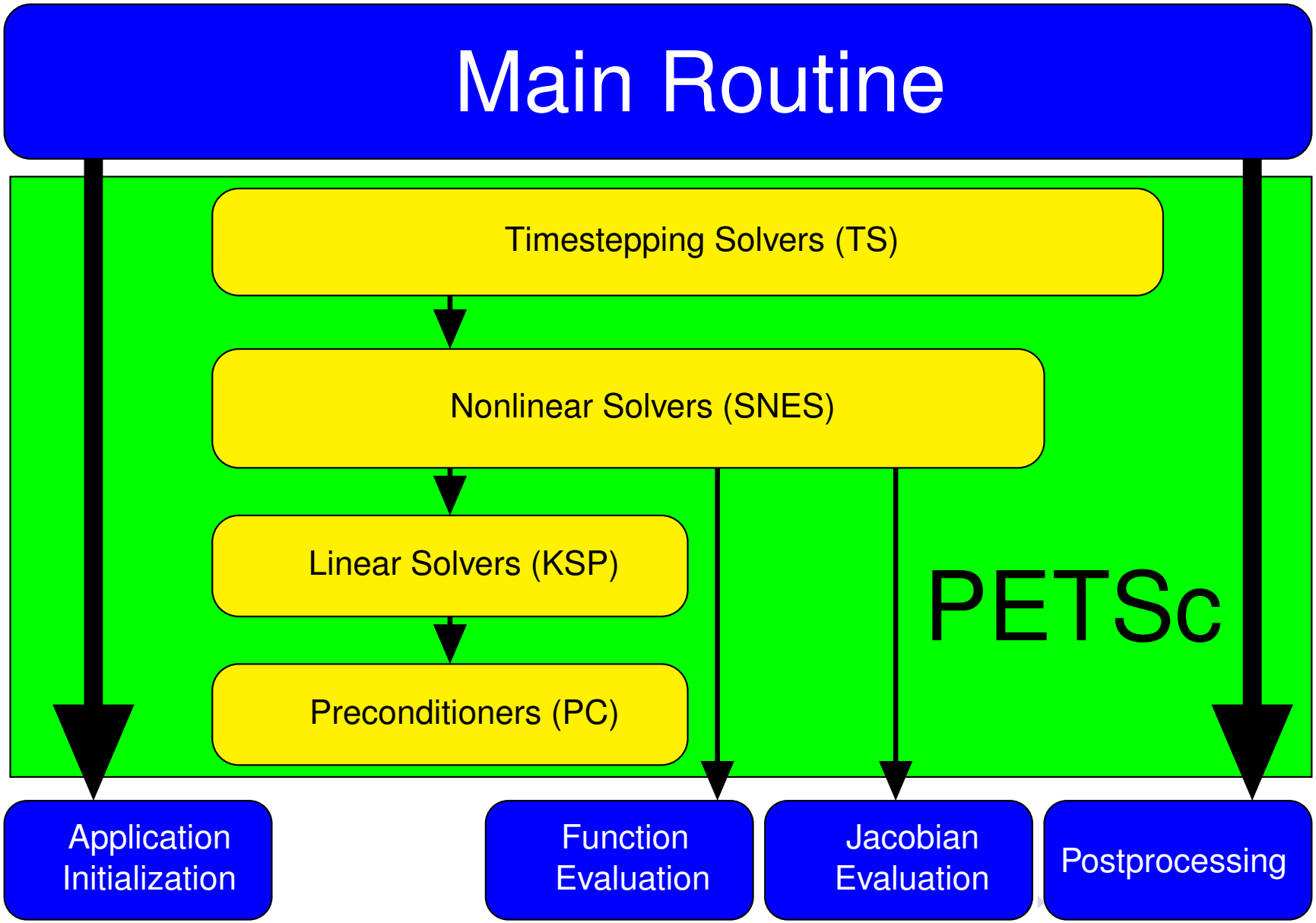
Outline

Getting More Help

- http://www.mcs.anl.gov/petsc
- Hyperlinked documentation
  - Manual
  - Manual pages for evey method
  - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov
- High profile users
  - David Keyes
  - Marc Spiegelman
  - Richard Katz
  - Brad Aagaard
  - Aron Ahmadia

Outline

Flow Control for a PETSc Application



SNES Paradigm

The SNES interface is based upon callback functions

- FormFunction(), set by SNESSetFunction()
- FormJacobian(), set by SNESSetJacobian()

When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Solver calls the **user's** function
- User function gets application state through the `ctx` variable
  - PETSc never sees application data

Topology Abstractions

- `DMDA`
  - Abstracts Cartesian grids in any dimension
  - Supports stencils, communication, reordering
  - Nice for simple finite differences
- `DMMesh`
  - Abstracts general topology in any dimension
  - Also supports partitioning, distribution, and global orders
  - Allows aribtrary element shapes and discretizations

Assembly Abstractions

- `DM`
  - Abstracts the logic of multilevel (multiphysics) methods
  - Manages allocation and assembly of local and global structures
  - Interfaces to `PCMG` solver
- `PetscSection`
  - Abstracts functions over a topology
  - Manages allocation and assembly of local and global structures
  - Will merge with `DM` somehow

Outline

Residual Evaluation

The **DM** interface is based upon *local* callback functions

- FormFunctionLocal()
- FormJacobianLocal()

Callbacks are registered using

- SNESSetDM(), TSSetDM()
- DMSNESSetFunctionLocal(), DMTSSetJacobianLocal()

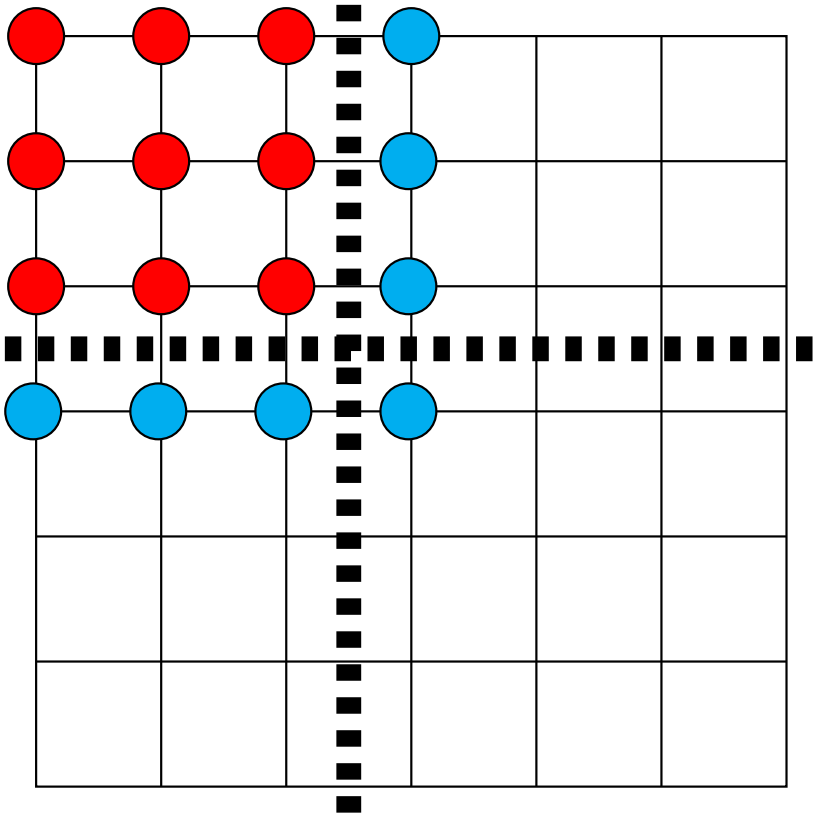
When PETSc needs to evaluate the nonlinear residual **F(x)**,

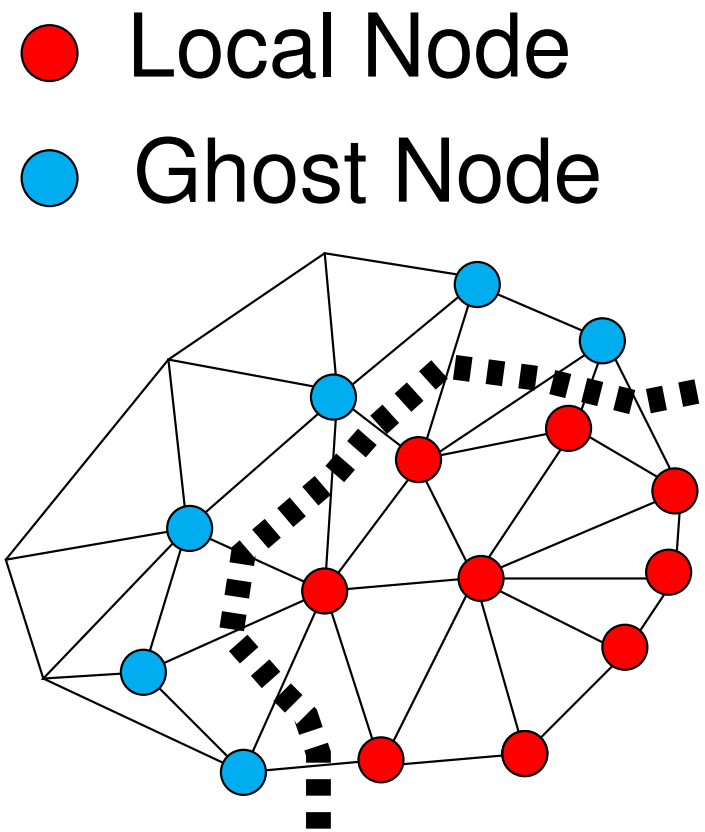
- Each process evaluates the local residual
- PETSc assembles the global residual automatically
  - Uses DMLocalToGlobal() method

Ghost Values

To evaluate a local function $f(x)$, each process requires

- its local portion of the vector x
- its ghost values, bordering portions of x owned by neighboring processes





DMDA Global Numberings

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

DMDA Global vs. Local Numbering

- **Global**: Each vertex has a unique id belongs on a unique process
- **Local**: Numbering includes vertices from neighboring processes
  - These are called ghost vertices

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

DMDA Local Function

User provided function calculates the nonlinear residual (in 2D)

```
(* lf )(DMDALocalInfo *info, PetscScalar**x, PetscScalar **r, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution (a multidimensional array)

`r`: The residual

`ctx`: The user context passed to `DMDASNESSetFunctionLocal()`

The local DMDA function is activated by calling

```
DMDASNESSetFunctionLocal(dm, INSERT_VALUES, lfunc, &ctx)
```

Bratu Residual Evaluation

$$\Delta u + \lambda e^u = 0$$

```
ResLocal(DMDALocalInfo *info, PetscScalar **x, PetscScalar **f, void *ctx)
for(j = info->ys; j < info->ys+info->ym; ++j) {
  for(i = info->xs; i < info->xs+info->xm; ++i) {
    u = x[j][i];
    if (i==0 || j==0 || i == M || j == N) {
      f[j][i] = 2.0*(hydhx+hxdhy)*u; continue;
    }
    u_xx     = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
    u_yy     = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
    f[j][i] = u_xx + u_yy - hx*hy*lambda*exp(u);
}}}
```

$PETSC_DIR/src/snes/examples/tutorials/ex5.c

DMDA Local Jacobian

User provided function calculates the Jacobian (in 2D)

(* ljac)(DMDALocalInfo *info, PetscScalar**x, Mat J, void *ctx)

`info`: All layout and numbering information

`x`: The current solution

`J`: The Jacobian

`ctx`: The user context passed to DASetLocalJacobian()

The local DMDA function is activated by calling

DMDASNESSetJacobianLocal(dm, ljac, &ctx)

A DMDA is more than a Mesh

A DMDA contains topology, geometry, and (sometimes) an implicit Q1 discretization.

It is used as a template to create

- Vectors (functions)
- Matrices (linear operators)

DMDA Vectors

- The **DMDA** object contains only layout (topology) information
  - All field data is contained in PETSc **Vecs**
- Global vectors are parallel
  - Each process stores a unique local portion
  - DMCreateGlobalVector(DM da, Vec *gvec)
- Local vectors are sequential (and usually temporary)
  - Each process stores its local portion plus ghost values
  - DMCreateLocalVector(DM da, Vec *lvec)
  - includes ghost and boundary values!

Updating Ghosts

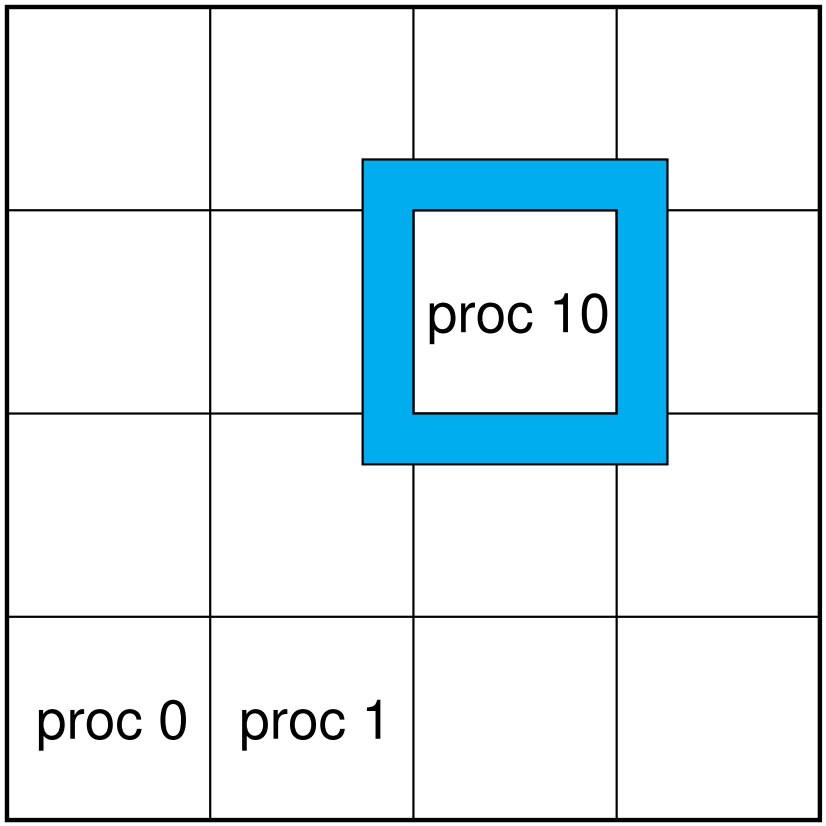
Two-step process enables overlapping computation and communication

- DMGlobalToLocalBegin(da, gvec, mode, lvec)
  - `gvec` provides the data
  - `mode` is either `INSERT_VALUES` or `ADD_VALUES`
  - `lvec` holds the local and ghost values
- DMGlobalToLocalEnd(da, gvec, mode, lvec)
  - Finishes the communication

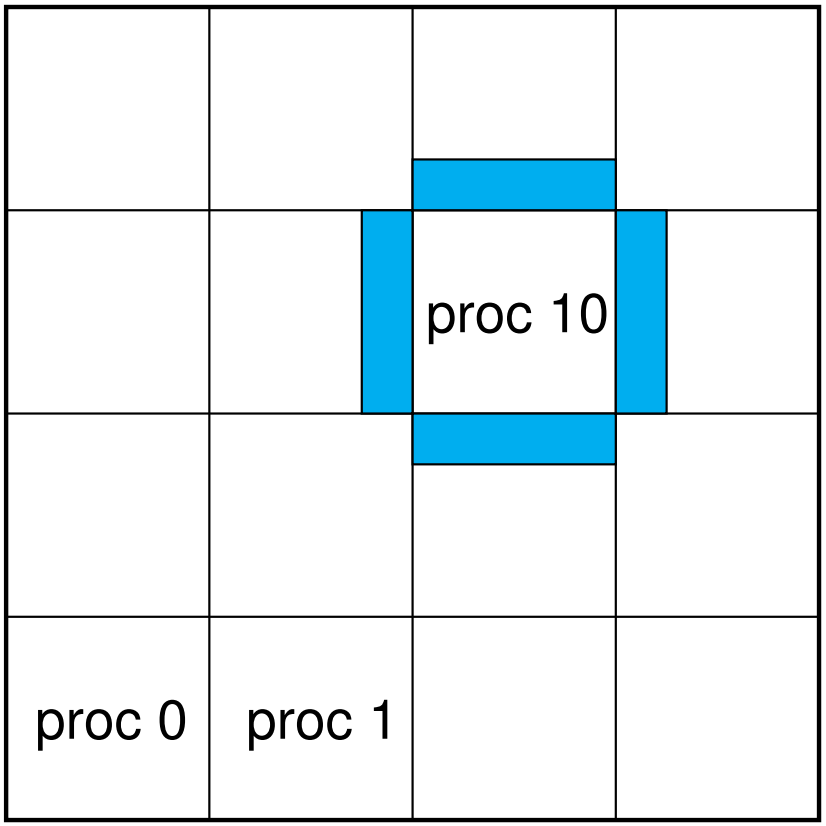
The process can be reversed with DALocalToGlobalBegin/End().

DMDA Stencils

Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

Setting Values on Regular Grids

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n, MatStencil idxn[],
                    PetscScalar values[], InsertMode mode)
```

- Each row or column is actually a **MatStencil**
  - This specifies grid coordinates and a component if necessary
  - Can imagine for unstructured grids, they are *vertices*
- The values are the same logically dense block in row/col

Outline

Mesh Paradigm

The **DMMesh** interface also uses *local* callback functions

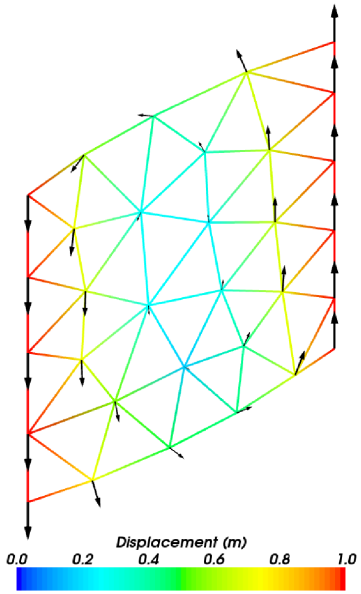
- maps between global Vec and local Vec
- Local vectors are structured using a **PetscSection**

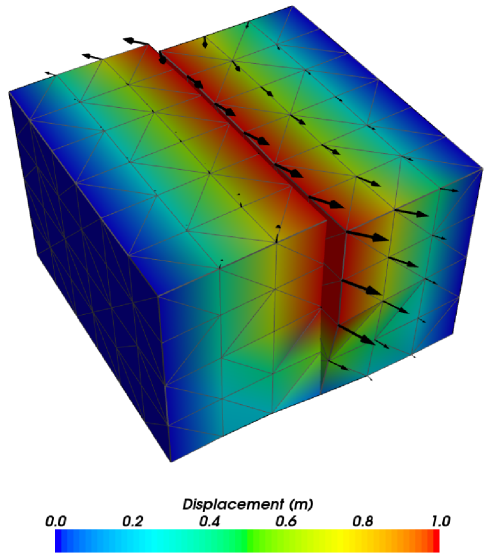
When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Each process evaluates the local residual for each element
- PETSc assembles the global residual automatically
  - `DMLocalToGlobal()` works just as in the structured case

Multiple Mesh Types

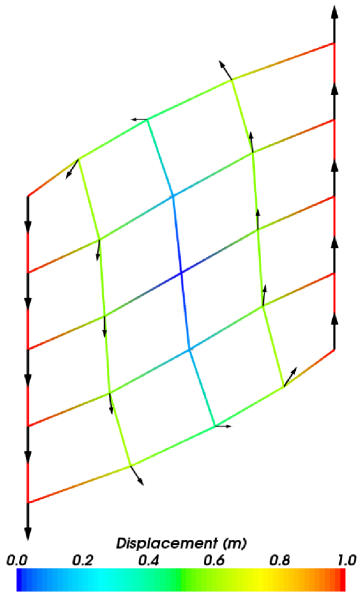
Triangular

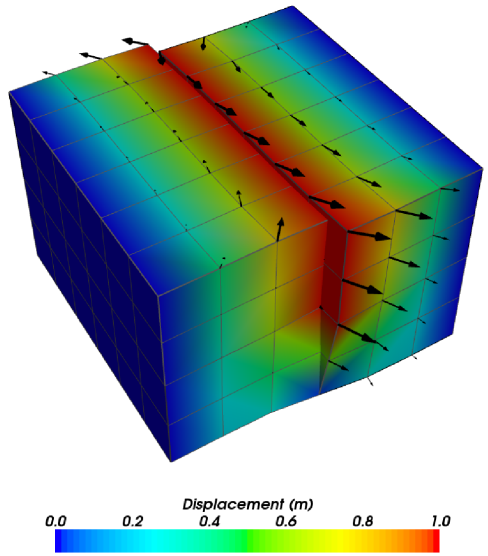




Tetrahedral

Rectangular

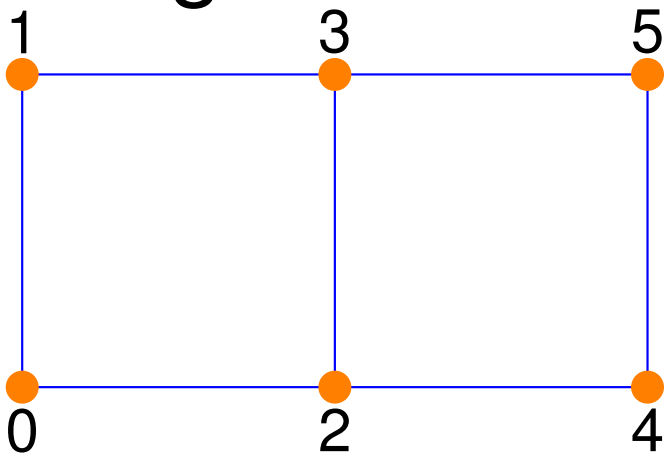




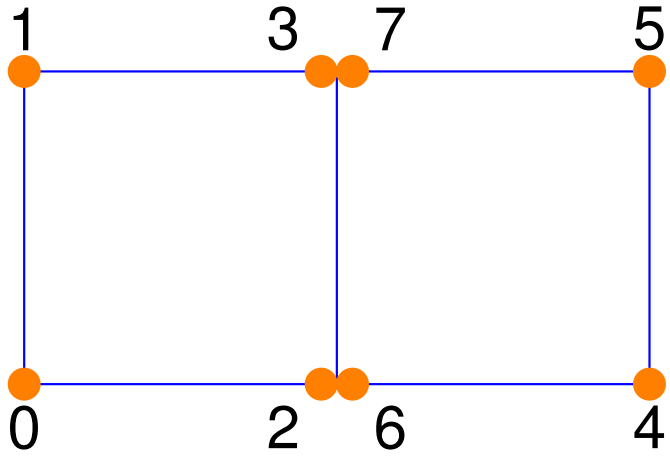
Hexahedral

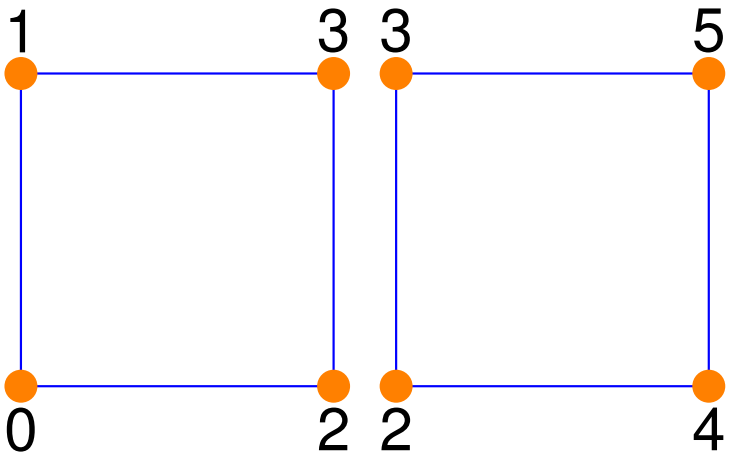
Cohesive Cells

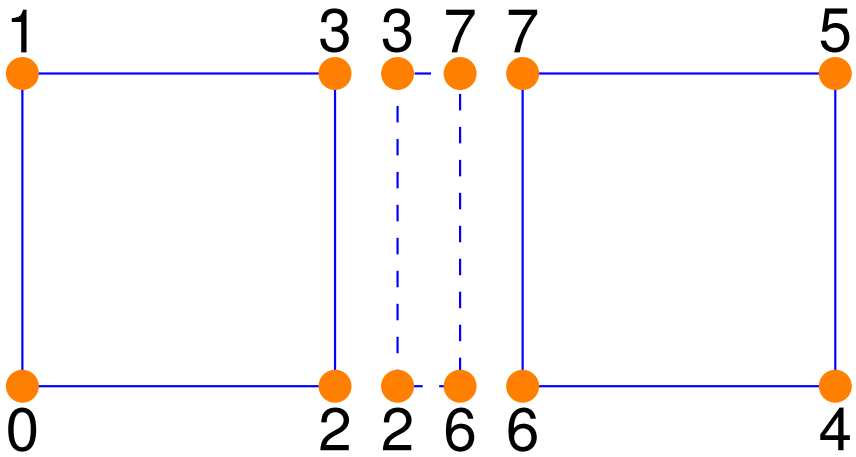
Original Mesh



Mesh with Cohesive Cell







Exploded view of meshes

Cohesive Cells

Cohesive cells are used to enforce slip conditions on a fault

- Demand complex mesh manipulation
  - We allow specification of only fault vertices
  - Must “sew” together on output
- Use Lagrange multipliers to enforce constraints
  - Forces illuminate physics
- Allow different fault constitutive models
  - Simplest is enforced slip
  - Now have fault constitutive models

Outline

Multigrid Paradigm

The **DM** interface uses the *local* callback functions to

- assemble global functions/operators from local pieces
- assemble functions/operators on coarse grids

Then **PCMG** organizes

- control flow for the multilevel solve, and
- projection and smoothing operators at each level.

DM Integration with SNES

- DM supplies global residual and Jacobian to SNES
  - User supplies local version to DM
  - The `Rhs_*()` and `Jac_*()` functions in the example
- Allows automatic parallelism
- Allows grid hierarchy
  - Enables multigrid once interpolation/restriction is defined
- Paradigm is developed in unstructured work
  - Solve needs scatter into contiguous global vectors (initial guess)
- Handle Neumann BC using `KSPSetNullSpace()`

Multigrid with DM

Allows multigrid with some simple command line options

- `-pc_type mg`, `-pc_mg_levels`
- `-pc_mg_type`, `-pc_mg_cycle_type`, `-pc_mg_galerkin`
- `-mg_levels_1_ksp_type`, `-mg_levels_1_pc_type`
- `-mg_coarse_ksp_type`, `-mg_coarse_pc_type`
- `-da_refine`, `-ksp_view`

Interface also works with GAMG and 3rd party packages like ML

Outline

MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
  - **VecScatter** and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
  - Generalizes `LocalToGlobal()` mapping
- is composable with ANY PETSc solver and preconditioner
  - This can be done recursively

MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
  - **VecScatter** and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
  - Generalizes `LocalToGlobal()` mapping
- is composable with ANY PETSc solver and preconditioner
  - This can be done recursively

FieldSplit provides the buildings blocks for multiphysics preconditioning.

MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
  - **VecScatter** and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
  - Generalizes `LocalToGlobal()` mapping
- is composable with ANY PETSc solver and preconditioner
  - This can be done recursively

Notice that this works in exactly the same manner as

- multiple resolutions (MG, FMM, Wavelets)
- multiple domains (Domain Decomposition)
- multiple dimensions (ADI)

Preconditioning

Several varieties of preconditioners can be supported:

- Block Jacobi or Block Gauss-Siedel
- Schur complement
- Block ILU (approximate coupling and Schur complement)
- Dave May's implementation of Elman-Wathen type PCs

which only require actions of individual operator blocks

Notice also that we may have any combination of

- "canned" PCs (ILU, AMG)
- PCs needing special information (MG, FMM)
- custom PCs (physics-based preconditioning, Born approximation)

since we have access to an algebraic interface

Outline

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

`http://www.fenics.org/fiat`

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

User can build arbitrary elements specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project, as is the PETSc Sieve module

FFC

FFC is a compiler for variational forms by Anders Logg.

Here is a mixed-form Poisson equation:

$$a((\tau, w), (\sigma, u)) = L((\tau, w)) \qquad \forall (\tau, w) \in V$$

where

$$\begin{aligned} a((\tau, w), (\sigma, u)) &= \int_\Omega \tau\sigma - \nabla \cdot \tau u + w \nabla \cdot u \, dx \\ L((\tau, w)) &= \int_\Omega wf \, dx \end{aligned}$$

FFC
Mixed Poisson

```
shape = "triangle"

BDM1 = FiniteElement("Brezzi-Douglas-Marini",shape,1)
DG0  = FiniteElement("Discontinuous Lagrange",shape,0)

element     = BDM1 + DG0
(tau, w)    = TestFunctions(element)
(sigma, u)  = TrialFunctions(element)

a = (dot(tau, sigma) - div(tau)*u + w*div(sigma))*dx

f = Function(DG0)
L = w*f*dx
```

FFC

Here is a discontinuous Galerkin formulation of the Poisson equation:

$$a(v,u) = L(v) \qquad \forall v \in V$$

where

$$\begin{aligned}
a(v,u) &= \int_\Omega \nabla u \cdot \nabla v \, dx \\
&+ \sum_S \int_S - < \nabla v > \cdot [[u]]_n - [[v]]_n \cdot < \nabla u > - (\alpha/h) vu \, dS \\
&+ \int_{\partial\Omega} -\nabla v \cdot [[u]]_n - [[v]]_n \cdot \nabla u - (\gamma/h) vu \, ds \\
L(v) &= \int_\Omega vf \, dx
\end{aligned}$$

FFC
DG Poisson

```
DG1 = FiniteElement("Discontinuous Lagrange",shape,1)
v = TestFunctions(DG1)
u = TrialFunctions(DG1)
f = Function(DG1)
g = Function(DG1)
n = FacetNormal("triangle")
h = MeshSize("triangle")
a = dot(grad(v), grad(u))*dx
  - dot(avg(grad(v)), jump(u, n))*dS
  - dot(jump(v, n), avg(grad(u)))*dS
  + alpha/h*dot(jump(v, n) + jump(u, n))*dS
  - dot(grad(v), jump(u, n))*ds
  - dot(jump(v, n), grad(u))*ds
  + gamma/h*v*u*ds
L = v*f*dx + v*g*ds
```

Outline

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations